JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 19 (2021) 159–164
Received: 27 April 2021
Accepted: 16 May 2021

Comparison of lightweight frameworks for Java by analyzing proprietary web applications

Porównanie lekkich szkieletów dla języka Java poprzez analizę autorskich aplikacji internetowych

Marek Pucek, Michał Błaszczyk*, Piotr Kopniak

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
There are many frameworks available in the IT environment that differ in performance, security, complexity, and speed. The user who wants to start working with the selected framework should know whether it will meet the architectural requirements and business assumptions. The aim of this work is to compare the performance and complexity of web applications built using different lightweight frameworks for the Java language. Spring Boot, Micronaut, Quarkus and Javalin frameworks will be compared. At the beginning of the work, the main problems related to the creation of modern applications were discussed. In the following sections, basic analysis of the complexity of the syntax and conventions of the selected skeletons was performed. Then, experiments were conducted to compare performance - response and build times and memory consumption during application development and use. A wide cross-section of efficiency has been obtained in selected lightweight framework usages. The prepared comparison can be used to select the appropriate framework for the project.

Keywords: Spring Boot; Micronaut; Quarkus; Javalin

Streszczenie
W środowisku IT dostępnych jest wiele szkieletów, które różnią się między sobą wydajnością, bezpieczeństwem, złożonością czy szybkością działania. Użytkownik, chcący zacząć pracę z wybranym szkieletem powinien wiedzieć, czy sprosta on wymaganiom architektonicznym oraz założeniom biznesowym. Celem niniejszej pracy jest porównanie wydajności oraz złożoności aplikacji internetowych zbudowanych z wykorzystaniem różnych lekkich szkieletów dla języka Java. Porównane zostaną szkielety Spring Boot, Micronaut, Quarkus oraz Javalin. Na początku pracy omówione zostały główne problemy związane z tworzeniem współczesnych aplikacji. W kolejnych częściach dokonano podstawowej analizy złożoności składni i konwencji wybranych szkieletów. Następnie wykonano eksperymenty mające na celu porównanie wydajności - czasy oraz zużycie pamięci podczas tworzenia i użytkowania aplikacji. Uzyskano szeroki przekrój efektywności w wybranych zastosowaniach lekkich szkieletów. Sporządzone porównanie może być wykorzystane do dobrania odpowiedniego szkieletu do projektu.

Słowa kluczowe: Spring Boot; Micronaut; Quarkus; Javalin

*Corresponding author
Email address: michal.blaszczyk@pollub.edu.pl (M. Błaszczyk)

1. Wstęp

Aplikacje internetowe w dzisiejszych czasach stanowią bardzo skuteczne rozwiązanie dla szerokiej gamy zadań biznesowych. Dzięki łatwości implementacji oraz możliwości dodawania kolejnych funkcjonalności stają się coraz częstszym wyborem wśród większości firm świadczących usługi swoim klientom. Zalety te powodują stopniowe wypieranie rozwiązań tradycyjnych takich jak aplikacje okienkowe.

W dzisiejszych czasach wybór szkieletu programistycznego dla aplikacji internetowej opartej o język Java wydaje się dosyć prosty - największą popularność od lat ma Spring Boot. Jednak istnieje jeszcze wiele innych lekkich szkieletów, których przeznaczeniem są zastosowania serwera aplikacji internetowej [1]. Celem niniejszej pracy jest porównanie oraz wskazanie wad i zalet kolejnych trzech - Micronauta, Javalina oraz Quarkusa.

Zestawienie zostało przedstawione z dwóch różnych punktów widzenia: programisty oraz użytkownika aplikacji. W pierwszym przypadku wzięte zostały pod uwagę aspekty takie jak analiza złożoności i wydajności kodu, konwencji przyjętych w wybranych szkieletach oraz sposobu komunikacji pomiędzy komponentami aplikacji. Dodatkowo porównano czasy kompilacji i uruchamiania aplikacji. Drugi punkt widzenia - użytkownika aplikacji - obejmuje głównie porównanie czasów przesyłania, pobierania oraz przetwarzania danych na serwerze.

2. Szkielety dla języka Java

2.1. Spring Boot

Spring Boot to szkielet aplikacyjny o otwartym kodzie źródłowym służący do tworzenia całej gamy aplikacji Java i umożliwiający uproszczenie procesu wytwarzania oprogramowania. Ma on na celu skrócenie długości kodu i zapewnienie najłatwiejszego sposobu tworzenia

aplikacji internetowych. Dzięki wbudowanym adnotacjom oraz autokonfiguracji Spring Boot pomaga stworzyć w krótkim czasie samodzielną aplikację z mniejszą lub prawie zerową konfiguracją. Oprócz tych podstawowych zalet można również wymienić inne cechy Spring Boota [2-3]:

- Umożliwia bardzo łatwe tworzenie aplikacji z użyciem języków Java lub Groovy.
- Skraca czas projektowania i zwiększa produktywność.
- Pozwala uniknąć pisania wielu linii standardowego kodu, adnotacji i konfiguracji XML.
- Bardzo łatwo jest zintegrować aplikację Spring Boot z ekosystemem Spring, w którego skład wchodzą Spring JDBC, Spring ORM, Spring Data, Spring Security, itp.
- Zapewnia wbudowane serwery HTTP, takie jak Tomcat, Jetty, itp., aby bardzo łatwo tworzyć i testować aplikacje internetowe [4].

2.2. Micronaut

Zwiększenie nacisku na tworzenie aplikacji opartych o mikroserwisy oraz rozwiązania chmurowe były głównym powodem dla stworzenia szkieletu Micronaut. W jego składni widać wiele podobieństw do szkieletu Spring Boot, takich jak podobne konwencje nazw czy praktycznie identyczne mechanizmy adnotacji. Jednak główną różnicą między Micronautem a Spring Bootem jest mechanizm wstrzykiwania zależności już w czasie kompilacji, twórcy Micronauta twierdzą, że dzięki temu udało im się uzyskać niższe czasy uruchamiania aplikacji. Omawiany szkielet wspiera trzy języki programowania Java, Kotlin oraz Groovy. Ze względu na silną optymalizację, twórcy szkieletu Micronaut postawili duży nacisk na integrację, z równie młodą technologią jaką jest GraalVM – uniwersalna lekka maszyna wirtualna [5-6].

2.3. Javalin

Javalin jest lekkim szkieletem dla języka Java oraz Kotlin. Podczas jego tworzenia położono duży nacisk na jak największe uproszczenie kodu. W przeciwieństwie do pozostałych omawianych szkieletów Javalin nie posiada adnotacji oraz mechanizmu refleksji, ma to zapewnić dużą szybkość działania aplikacji przy jednoczesnym oszczędzaniu pamięci i zasobów procesora. W celu uruchomienia aplikacji należy utworzyć obiekt o typie Javalin i wywołać metodę uruchamiającą ją na wybranym porcie. Konfigurację oraz pozostałe aspekty aplikacji wprowadza się edytując stworzony obiekt [7].

2.4. Quarkus

Podobnie jak w przypadku Micronauta szkielet Quarkus jest odpowiedzią na rosnące zapotrzebowanie na aplikacje oparte o mikroserwisy oraz aplikacje chmurowe. Został opracowany przez RedHat i jest przeznaczony do tworzenia aplikacji, które mają być wdrażane w chmurze, natywnie obsługując Kubernetes i umożliwiając łatwe budowanie lekkich aplikacji pod względem dostarczanego rozmiaru i wykorzystania pamięci [8-9]. Pozostałymi, ale nie mniej ważnymi zaletami Quarkusa są również:

- ujednolicona konfiguracja przechowywana w jednym pliku,
- szybkie przeładowanie konfiguracji – wszystkie zmiany w konfiguracji aplikują się wraz z odświeżeniem uruchomionej aplikacji,
- uproszczony kod dla 80% typowych zastosowań, elastyczny dla 20% [10],
- łatwe generowanie natywnych plików wykonywalnych.

3. Cel i obiekt badań

Celem badań jest analiza kodu i wydajności czterech aplikacji stworzonych za pomocą omawianych szkieletów aplikacyjnych, a następnie wskazanie różnic, wad oraz zalet poszczególnych frameworków.

4. Przegląd narzędzi do budowy aplikacji

Do napisania kolejnych aplikacji użyto następujących narzędzi:

- JDK 11 – zestaw narzędzi programistycznych Javy,
- PostgreSQL – baza danych użyta w testowej aplikacji internetowej,
- Maven – narzędzie automatyzujące budowę aplikacji – użytą w przypadku Spring Boota oraz Quarkusa,
- Gradle – narzędzie automatyzujące budowę aplikacji – użytą w przypadku Micronauta i Javalina,
- Tomcat – serwer WWW i kontener aplikacyjny – wykorzystany do tworzenia większości aplikacji oraz jako środowisko produkcyjne,
- Jetty – serwer aplikacyjny – używany przy tworzeniu aplikacji Javalina,
- JMeter – narzędzie testowe wykorzystane przy testach punktów dostępowych,
- Jenkins – narzędzie do automatycznego budowania i uruchamiania projektów.

5. Zaimplementowane aplikacje

Wszystkie z zaimplementowanych aplikacji posiadają podobną architekturę. Przykładowymi klasami na podstawie, których można dokonać analizy są:

- klasa główna aplikacji,
- domeny,
- kontroler,
- repozytorium.

Klasa główna w przypadku Spring Boota, Micronauta oraz Quarkusa wygląda bardzo podobnie, składa się ona z adnotacji specyficznej dla danego szkieletu oraz jednej statycznej metody uruchamiającej aplikacje. W przypadku Javalina główna klasa jest o wiele bardziej rozbudowana. W metodzie main znajduje się inicjalizacja obiektu o typie Javalin oraz z jego konfiguracja.

W przypadku klas encyjnych (domenowych), ponownie w szkieletach Spring Boot, Micronaut oraz Quarkus kod klas wygląda bardzo podobnie. Ponad nagłówkami klas znajdują się adnotacje wskazujące nazwy tabel w bazie danych odnoszących się do danej domeny. Nad poszczególnymi polami znajdują się ad-

notacje zawierające nazwy kolumn oraz właściwości takie jak np. id lub relacje między tabelami. W przypadku Javalina domeny są jedynie klasami ze wszystkimi polami domeny. Częścią wspólną wszystkich klas są dwa konstruktory, jeden pusty – wykorzystywany do serializacji, drugi posiadający pola klasy.

Kontrolery w aplikacjach opartych o szkielety Spring Boot, Micronaut oraz Quarkus także są bardzo do siebie podobne. Różnice polegają jedynie na nazwach adnotacji. Kontroler aplikacji opartej o szkielet Javalin musi posiadać obiekt o typie Javalin utworzony w klasie głównej, następnie przy pomocy metod wykonywanych na tym obiekcie dodawane są kolejne punkty dostępowe.

Ze względu na to, że aplikacje nie posiadają złożonych funkcjonalności operujących na bazie danych, w przypadku aplikacji opartych o szkielety Spring Boot, Micronaut oraz Quarkus możliwe było zastosowanie stylu architektonicznego CRUD. Umożliwia to używanie domyślnych metod operujących na bazie danych. Możliwe jest także tworzenie własnych bardziej skomplikowanych metod używając sugerowanej przez szkielety składni - należy zauważyć, że składnie różnią się między szkieletami. Dzięki temu możliwe było zaimplementowanie metody wyszukującej obiekty posiadające swoje id w podanym przedziale. Szkielet Javalin nie posiada wymienionych wyżej mechanizmów, z tego powodu należało wykorzystać zapytania SQL uzupełniane o wymagane parametry.

6. Metodyki testów

Badanie stara się odpowiedzieć na pytanie: który z frameworków jest najlepszym wyborem przy tworzeniu aplikacji internetowej. Istnieje wiele metod porównywania szkieletów aplikacyjnych. Jednym z nich są testy wydajnościowe przeprowadzone na aplikacji stworzonej z użyciem badanych frameworków. Aby testy były jeszcze bardziej wiarygodne zdecydowano się na przeprowadzenie ich na dwóch maszynach z różnymi systemami operacyjnymi. Ich specyfikacje przedstawiono w Tabeli 1 i 2.

Tabela 1: Parametry maszyny 1

Maszyna nr 1	
Procesor	Intel Core i7 3770K, 4 rdzenie, 8 wątków, taktowanie 3,50 GHz (CPU Mark – 6422)
Pamięć RAM	24,0 GB
Rodzaj dysku	SSD
System operacyjny	Manjaro Linux 20.1.2

Tabela 2: Parametry maszyny 2

Maszyna nr 2	
Procesor	AMD Ryzen 5 2600, 6 rdzeni, 12 wątków, taktowanie 3,40 GHz (CPU Mark – 13219)
Pamięć RAM	32,0 GB
Rodzaj dysku	SSD
System operacyjny	Windows 10 Education 10.0.18363

Dla każdej z omówionych wcześniej aplikacji przeprowadzono cztery testy mające na celu zbadać wydajność każdego ze szkieletów.

6.1. Test uruchamiania aplikacji w środowisku produkcyjnym

Ze względu na rosnącą popularność praktyki rozwoju oprogramowania opierającego się o ciągłą integrację (continuous integration) pierwszy z testów polegał na zmierzeniu czasu uruchamiania aplikacji w środowisku produkcyjnym. Każda z aplikacji została zbudowana oraz umieszczona na serwerze aplikacyjnym. Zdecydowano się na powtórzenie testu 50 razy dla każdej z nich. Test zawierał następujące kroki:

- Wyczyszczenie folderu roboczego (workspace) Jenkinsa,
- Pobranie projektu aplikacji do folderu Jenkinsa,
- Wywołanie zadań narzędzi automatyzujących: clean - czyszczący projekt oraz package/war kompilujący oraz pakujący projekt do paczki dystrybucyjnej,
- Umieszczenie paczki na serwerze Tomcat.

6.2. Test uruchamiania aplikacji w środowisku programistycznym

Podczas tworzenia dużych projektów jednym z bardziej czasochłonnych procesów jest budowanie aplikacji na lokalnym środowisku programistycznym. Problem ten został zaadresowany w drugim z testów. Wszystkie projekty aplikacji zostały uruchomione 20 razy na każdej z testowych maszyn. Proces uruchomienia obejmował zbudowanie aplikacji oraz umieszczenie jej na stosownym serwerze aplikacyjnym. W przypadku Spring Boota oraz Micronauta projekty zostały zbudowane oraz umieszczone na zintegrowanym z IDE serwerze Tomcat, w przypadku Quarkusa projekt zostaje przebudowany i uruchomiony na serwerze wbudowanym Netty. Ze względu na domyślną konfigurację oraz zalecenia dokumentacji szkielet Javalin został zbudowany i umieszczony na serwerze Jetty.

6.3. Test zużycia pamięci oraz procesora podczas działania aplikacji

Podczas wydawania aplikacji do ogólnego użytku ważne jest, aby jak największa liczba urządzeń była w stanie jej używać. Dużą znaczenie ma przy tym obciążenie procesora oraz ilość pamięci zużywanej przez wirtualną maszynę Javy, która przy złym zarządzaniu może przekroczyć możliwości urządzenia i uniemożliwić działanie programu. W kolejnym teście przetestowano zużycie pamięci oraz procesora przez każdą z aplikacji pod wybranym obciążeniem. Wszystkie aplikacje zostały zbudowane oraz umieszczone na serwerze aplikacyjnym. Pierwsza część testu polegała na zbadaniu zużycia w stanie bezczynności. Dla każdego szkieletu wykonano pomiar trwający 10 minut. Przez kolejne 5 minut aplikacje były poddane lekkiemu obciążeniu w postaci zapytań wysyłanych przez jednego użytkownika, pobierających 50 rekordów z bazy danych. W ostatniej fazie testu przez 5 minut wykonano silne obciążenie w posta-

ci zapytań wysyłanych przez 20 użytkowników pobierających 3 miliony rekordów z bazy danych.

6.4. Test szybkości punktów dostępowych

Ostatnim, ale nie mniej ważnym kryterium wziętym pod uwagę w testach jest szybkość komunikacji klienta z serwerem. W dzisiejszych czasach popularne aplikacje muszą być gotowe na odpowiedź na setki a nawet tysiące zapytań w ciągu sekundy. W ostatnim z testów sprawdzono jak szybko omawiane aplikacje odpowiedzą na dużą liczbę zapytań. Wszystkie aplikacje zostały zbudowane oraz umieszczone na serwerze aplikacyjnym. Pierwsza część testu polegała na zmierzeniu czasów odpowiedzi na zapytanie GET. Dla każdego szkieletu wykonano obciążenie w postaci 8000 zapytań, każde pobierające 1000 rekordów z bazy danych. Następnie wysłano 8000 zapytań POST, z których każde dodawało 200 rekordów do bazy danych. Na końcu obliczono jaki był łączny czas wysłania wszystkich zapytań GET oraz POST.

7. Analiza otrzymanych wyników

7.1. Test uruchamiania aplikacji w środowisku produkcyjnym

W pierwszym z testów zestawiono średnie czasy kolejnych kroków budowania aplikacji.

Tabela 3: Średnie czasy kolejnych kroków budowania aplikacji na poszczególnych maszynach

	Kompilacja		Pakowanie		Umieszczanie	
	Masz. 1	Masz. 2	Masz. 1	Masz. 2	Masz. 1	Masz. 2
Spring Boot	1,201 s	1,367 s	1,526 s	2,206 s	5,857 s	12,195 s
Micronaut	1,776 s	2,14 s	0,008 s	0,005 s	4,983 s	10,412 s
Javalin	0,24 s	0,288 s	0,004 s	0,004 s	1,892 s	2,05 s
Quarkus	1,158 s	1,517 s	0,135 s	0,169 s	6,28 s	7,721 s

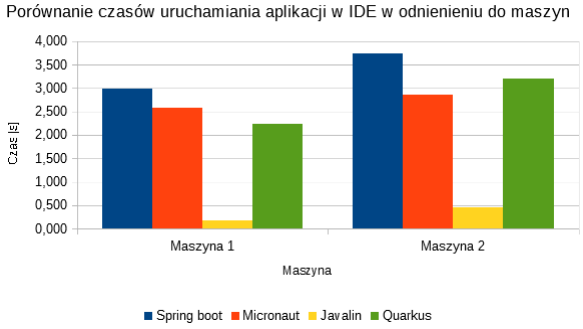
Przedstawione w Tabeli 3 czasy poszczególnych kroków budowania projektu znacznie różnią się między sobą. Aplikacja wykorzystująca Javalin z racji braku skomplikowanych mechanizmów obecnych w innych szkieletach kompiluje się znacznie szybciej od reszty. Quarkus oraz Spring Boot posiadają bardzo zbliżone wyniki. Micronaut ze względu na jego proces wstrzykiwania zależności w czasie kompilacji wykonywał ten proces najdłużej. W przypadku pakowania aplikacji do paczki dystrybucyjnej w przypadku Javalin oraz Micronauta, etap ten wykonywał się on praktycznie natychmiastowo, a nieco wolniejszy okazał się Quarkus. Jedynym szkieletem aplikacyjnym w którym czas był odczuwalny jest Spring Boot. W procesie umieszczania paczki dystrybucyjnej na serwerze najszybszym frameworkiem ponownie jest Javalin, następnie Quarkus - ze względu na swoją budowę uruchamiany na własnym serwerze. Dzięki procesom wykonywanym podczas kompilacji ostatni krok Micronaut wykonał szybciej od Spring Boota.

7.2. Test uruchamiania aplikacji w środowisku programistycznym

W kolejnym teście porównano średnie czasy uruchamiania projektów w zależności od maszyny.

Tabela 4: Porównanie czasów uruchamiania aplikacji między maszynami

	Spring Boot	Micronaut	Javalin	Quarkus
Maszyna 1	2,976 s	2,582 s	0,176 s	2,244 s
Maszyna 2	3,745 s	2,852 s	0,454 s	3,199 s



Rysunek 1: Porównanie czasów uruchamiania aplikacji między maszynami.

Analizując przedstawione w Tabeli 4 średnie czasy uruchamiania aplikacji, zauważyć można dosyć dużą różnicę w czasach pomiędzy Javalinem, a resztą szkieletów. Wynika to z faktu zmiany serwera aplikacyjnego na rekomendowany Jetty – dużo lżejszy w porównaniu do Tomcat, drugim powodem jest budowa samego Javalina mniej skomplikowanego od pozostałych frameworków. Kolejne dwa szkielety z porównywalnymi czasami to Micronaut oraz Quarkus. Najwolniejszy ponownie okazał się Spring Boot uruchamiający się powyżej trzech sekund. Dla lepszego zobrazowania różnic w czasach użyto wykresu kolumnowego przedstawionego na Rysunku 1.

7.3. Test zużycia pamięci oraz procesora podczas działania aplikacji

Następnie przy użyciu narzędzi Prometheus, Grafana oraz VisualVM zbadano zużycie procesora oraz pamięci dla każdej z aplikacji w stanie spoczynku, przy małym oraz dużym obciążeniu.

Analizując wyniki przedstawione w Tabeli 5 można stwierdzić, że nie ma większych różnic w zużyciu zasobów przez omawiane aplikacje w stanie spoczynku. Dla żadnej z nich średnie zużycie procesora nie przekroczyło 4%. Mimo wszystko najlepiej pod tym kątem prezentowała się aplikacja napisana z wykorzystaniem szkieletu Javalin, dla której zużycie procesora utrzymywało się najbliżej 0%. W przypadku tego frameworka średnie

zużycie pamięci również było najwyższe i wynosiło jedynie 30MB na maszynie 2.

Tabela 5: Średnie zużycie zasobów w stanie spoczynku

	Maszyna 1		Maszyna 2	
	CPU	Pamięć (MB)	CPU	Pamięć (MB)
Spring Boot	2%	282	2%	138
Micronaut	4%	301	3%	231
Quarkus	1,5%	184	4%	163
Javalin	0%	50	0%	30

Tabela 6: Średnie zużycie zasobów przy małym obciążeniu

	Maszyna 1		Maszyna 2	
	CPU	Pamięć (MB)	CPU	Pamięć (MB)
Spring Boot	68%	350	58%	350
Micronaut	75%	360	28%	250
Quarkus	68%	350	58%	350
Javalin	1%	325	3%	250

Dla badań przeprowadzonych przy lekkim obciążeniu wyniki okazały się o wiele ciekawsze. Zostały one zaprezentowane w Tabeli 6. Pojawiła się bowiem zauważalna rozbieżność w użyciu procesora między używanymi maszynami. W większości przypadków było ono niższe na maszynie 2, w przypadku Micronauta aż o 47 p.p. Wyjątkiem był ponownie Javalin, dla którego maszyna 1 mogła pochwalić się lepszym rezultatem. W przypadku tego szkieletu zużycie pamięci na obu maszynach było również najmniejsze, ale różnica w stosunku do pozostałych frameworków nie była aż tak znacząca jak w przypadku pierwszego testu.

Tabela 7: Średnie zużycie zasobów przy dużym obciążeniu

	Maszyna 1		Maszyna 2	
	CPU	Pamięć (MB)	CPU	Pamięć (MB)
Spring Boot	100%	200	98%	380
Micronaut	21%	350	27%	210
Quarkus	99%	700	99%	300
Javalin	8%	130	9%	130

Tabela 7 przedstawia wyniki uzyskane przy największym obciążeniu. Dużym zaskoczeniem okazały się aplikacje napisane z użyciem Micronaut oraz Javalin, dla których wykorzystanie procesora było bardzo małe (około 27% w przypadku Micronauta i 9% w przypadku Javalin) w porównaniu do pozostałych, gdzie zużycie wyniosło prawie 100%. Jeżeli chodzi o wykorzystanie pamięci to tak jak w przypadku pierwszego testu było ono wyraźnie niższe dla frameworka Javalin – dla pozostałych wyniki były zbliżone do tych przy małym obciążeniu. Wyjątkiem okazał się szkielet Quarkus, dla którego zużycie pamięciu było kilkukrotnie większe od jego konkurentów.

7.4. Test szybkości punktów dostępowych

Tabela 8: Porównanie czasów odpowiedzi

	GET		POST	
	Maszyna 1	Maszyna 2	Maszyna 1	Maszyna 2
Spring Boot	1398,3 s	2126,1 s	5085,5 s	4024,2 s
Micronaut	1849,8 s	3955,4 s	6855,5 s	5406 s
Quarkus	1509,5 s	2401,7 s	5550,2 s	4660,9 s
Javalin	467,4 s	1430,6 s	886,5 s	1586,4 s

Analizując wyniki przedstawione w Tabeli 8 można stwierdzić, że zdecydowanie najkrótsze czasy odpowiedzi serwera na obu maszynach uzyskała aplikacja napisana przy użyciu szkieletu Javalin. Był on kilkukrotnie krótszy, szczególnie w przypadku zapytania POST niż w przypadku Spring Boota, który zajął drugą lokatę w tym porównaniu. Co ciekawe najgorzej w tym zestawieniu wypadł Micronaut, który z założenia jest przeznaczony do tworzenia architektur mikroserwisowych, w których czas komunikacji z serwerem jest szczególnie istotny.

8. Wnioski

Celem niniejszej pracy było porównanie lekkich szkieletów programistycznych biorąc pod uwagę ich kluczowe elementy takie jak wydajność, szybkość oraz składnię.

Podczas implementacji aplikacji w poszczególnych szkieletach wskazano różnice oraz podobieństwa w konwencjach i rozwiązaniach w nich zaimplementowanych. Dokonano także porównania narzędzi dedykowanych do utworzenia podstawowego projektu. Tworzenie projektu pozwoliło na empiryczne sprawdzenie możliwości porównywanych frameworków. Na tej podstawie można zauważyć bardzo duże podobieństwo trzech szkieletów Spring Boota, Micronauta oraz Quarkusa – przedstawiają one podobne podejścia do konwencji i stylu tworzenia aplikacji. Spośród tych trzech frameworków z punktu widzenia programisty wyróżnia się szkielet Quarkus. Posiada on wiele przydatnych cech takich jak „hot deployment" czy specjalny tryb deweloperski. Całkowicie inne podejście do tematu implementacji aplikacji internetowej pokazuje Javalin. W swojej konstrukcji opiera się na prostocie i posiada jedynie podstawowe funkcjonalności. Niestety wadą tego rozwiązania jest to, że programista musi tworzyć własne

rozwiązania od podstaw lub adaptować zewnętrzne biblioteki.

Przeprowadzone badania wyraźnie wskazują aplikację opartą o szkielet Javalin jako najwydajniejszą. Pozostałe aplikacje wykonywały się znacznie wolniej oraz potrzebowały dużo więcej zasobów do swojej pracy. Najgorsze wyniki podczas testów osiągnął szkielet Spring Boot jednak jest on szeroko stosowany ze względu na dużą funkcjonalności i rozbudowane środowisko dodatków, wspieranych standardów oraz łatwo dostępne wsparcie społeczności.

Literatura

[1] D. Curie, J. Jaison, J. Yadav, J. Fiona, Analysis on Web Frameworks. Journal of Physics: Conference Series, 1362 (2019) 012114 doi:10.1088/1742-6596/1362/1/012114

[2] R. Rakshith Rao, S.R. Swamy, Review on Spring Boot and Spring Webflux for Reactive Web Development, International Research Journal of Engineering and Technology, 7(04) (2020) 3843-3837.

[3] Opis odwrócenia sterowanie w szkielecie Spring Boot, https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring , [28.02.2021].

[4] Oficjalna dokumentacja szkieletu Spring Boot, https://docs.spring.io/spring-framework/docs/3.0.0.M3/reference/html/ch01s02.html , [28.02.2021].

[5] Oficjalna dokumentacja szkieletu Micronaut, https://micronaut.io/docs, [28.02.2021].

[6] Wprowadzenie do szkieletu Micronaut, https://www.baeldung.com/micronaut , [28.02.2021].

[7] Oficjalna dokumentacja szkieletu Javalin, https://javalin.io/documentation , [28.02.2021].

[8] M. Šipek, D. Muharemagić, B. Mihaljević, A. Radovan, Enhancing Performance of Cloud-based Software Applications with GraalVM and Quarkus, 43rd International Convention on Information, Communication and Electronic Technology, (MIPRO) (2020) 1746-1751, doi: 10.23919/MIPRO48935.2020.9245290.

[9] Przegląd funkcjonalności szkieletu Quarkus, https://www.redhat.com/en/topics/cloud-native-apps/what-is-quarkus , [28.02.2021].

[10] Oficjalna dokumentacja szkieletu Quarkus, https://quarkus.io/ , [28.02.2021].